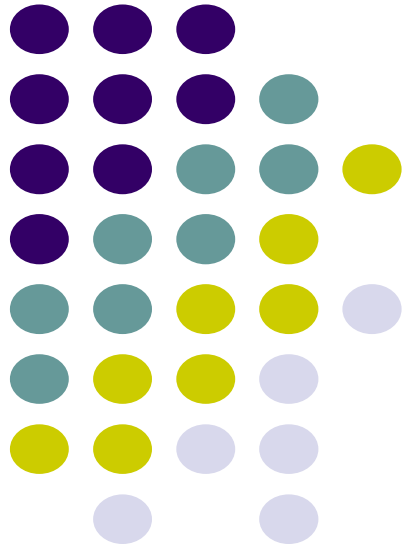
The Basics of UNIX/Linux

11-2. Arrays and Pointer. Part 2

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU



This file is based on slides from Justin Hsia.

datalab

data science laboratory

Lecture Outline

- **Pointers & Pointer Arithmetic**
- Pointers as Parameters

Box-and-Arrow Diagrams (1/4)

boxarrow.c

```
int main(int argc, char** argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

  return 0;
}
```

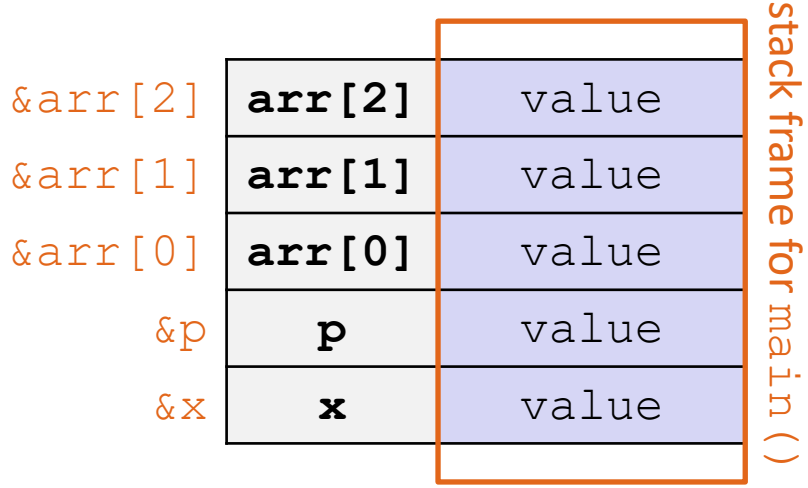
address	**name**	value

Box-and-Arrow Diagrams (2/4)

boxarrow.c

```
int main(int argc, char** argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

  return 0;
}
```

address	**name**	value



Box-and-Arrow Diagrams (3/4)

boxarrow.c

```
int main(int argc, char** argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

  return 0;
}
```

address	**name**	value

&arr[2]	**arr[2]**	4
&arr[1]	**arr[1]**	3
&arr[0]	**arr[0]**	2
&p	**p**	&arr[1]
&x	**x**	1

Box-and-Arrow Diagrams (4/4)

boxarrow.c

```
int main(int argc, char** argv) {
  int x = 1;
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];

  printf("&x: %p;  x: %d\n", &x, x);
  printf("&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);
  printf("&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);

  return 0;
}
```

address	**name**	value

p: get addr

*p: get data at addr (follow arrow)

0x7fff…78	**arr[2]**	4
0x7fff…74	**arr[1]**	3
0x7fff…70	**arr[0]**	2
0x7fff…68	**p**	0x7fff…74
0x7fff…64	**x**	1

Pointer Arithmetic

- Pointers are *typed*
  - Tells the compiler the size of the data you are pointing to
  - Exception: `void*` is a generic pointer (*i.e.* a placeholder)

- Pointer arithmetic is scaled by `sizeof(*p)`
  - Works nicely for arrays
  - Does not work on `void*`, since `void` doesn't have a size!

- Valid pointer arithmetic:
  - Add/subtract an integer to a pointer
  - Subtract two pointers (within stack frame or malloc block)
  - Compare pointers (<, <=, ==, !=, >, >=), including `NULL`

Practice Question

boxarrow2.c

```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return 0;
}
```

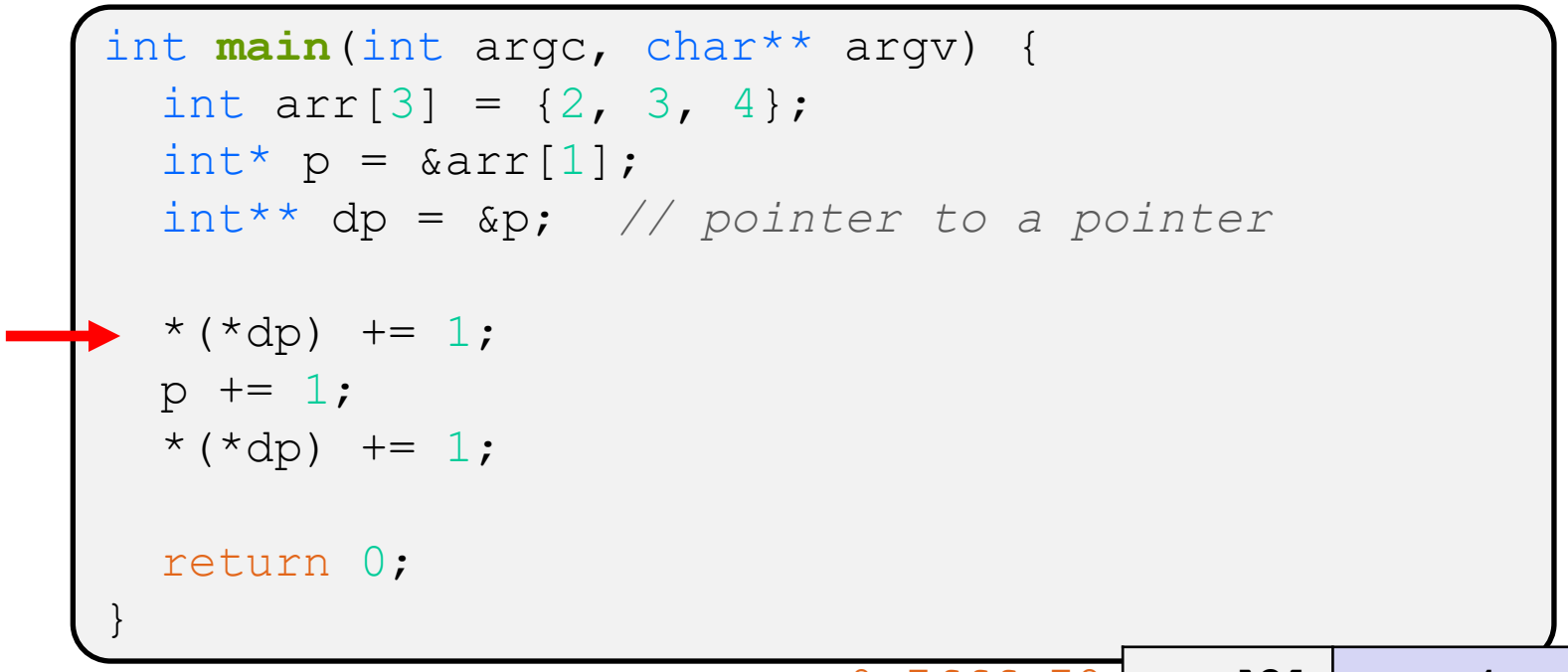
At this point in the code, what values are stored in `arr[]`?

address	name	value
0x7fff…78	arr[2]	4
0x7fff…74	arr[1]	3
0x7fff…70	arr[0]	2
0x7fff…68	p	0x7fff…74
0x7fff…60	dp	0x7fff…68

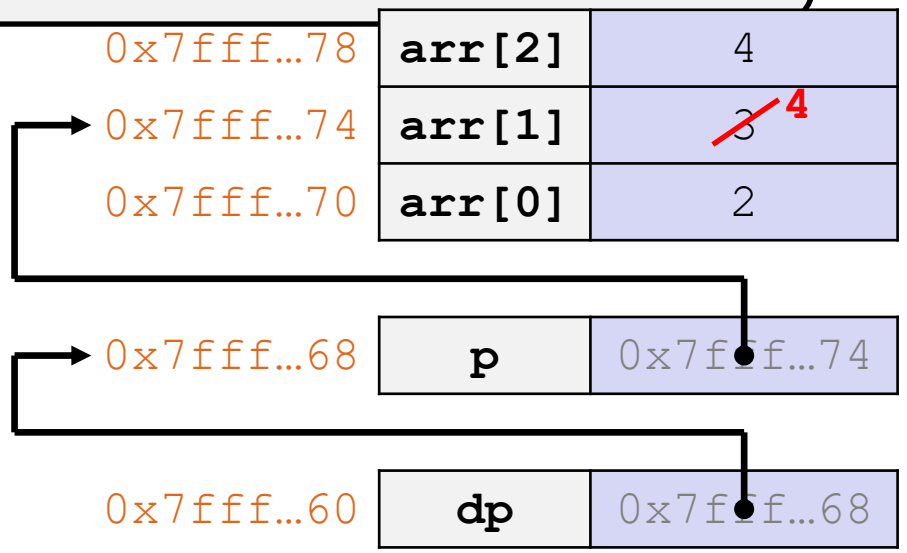
Practice Solution (1/5)

Note: arrow points to *next* instruction to be executed.

boxarrow2.c



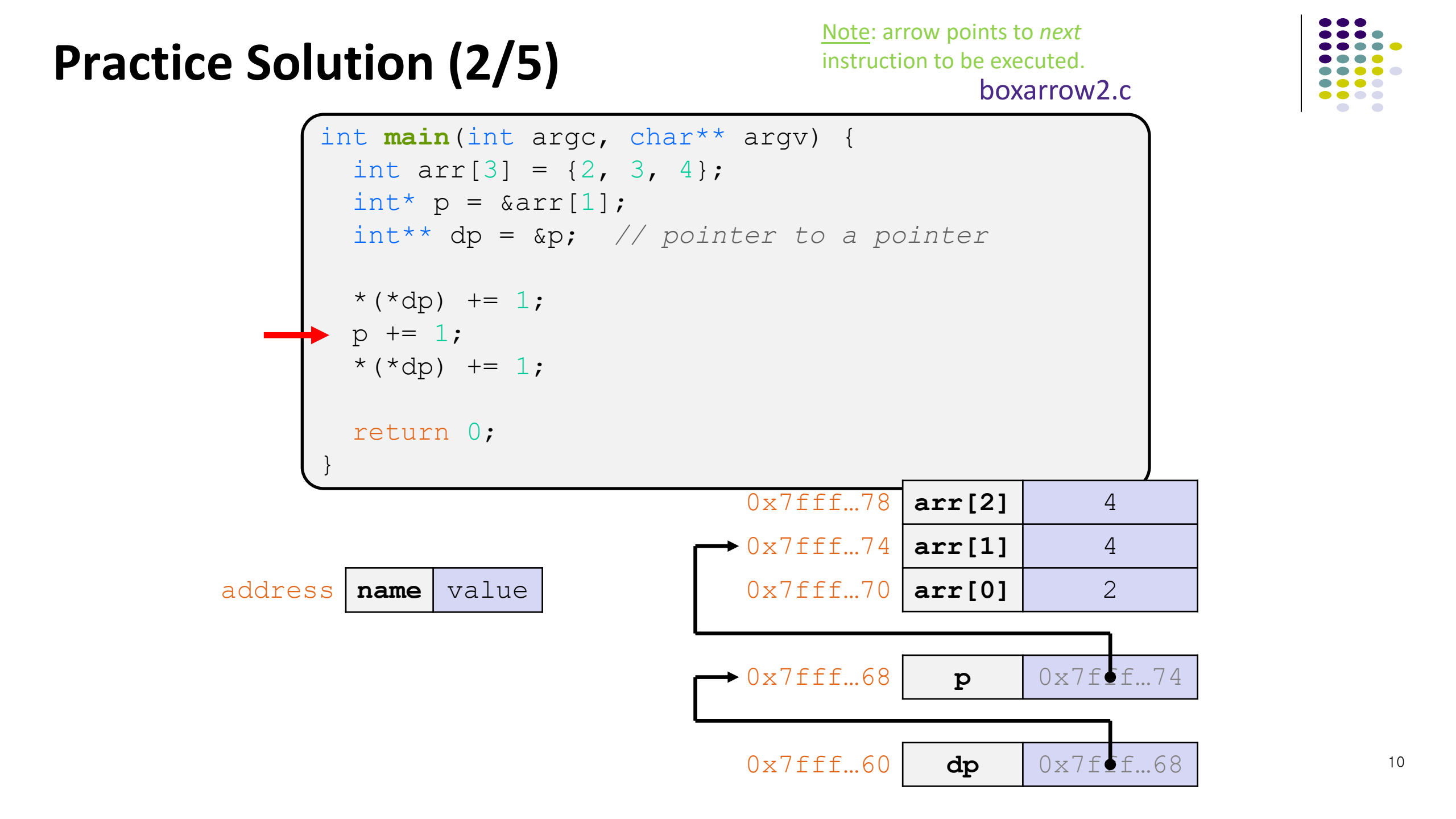
```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return 0;
}
```



Practice Solution (2/5)

Note: arrow points to *next* instruction to be executed.

boxarrow2.c

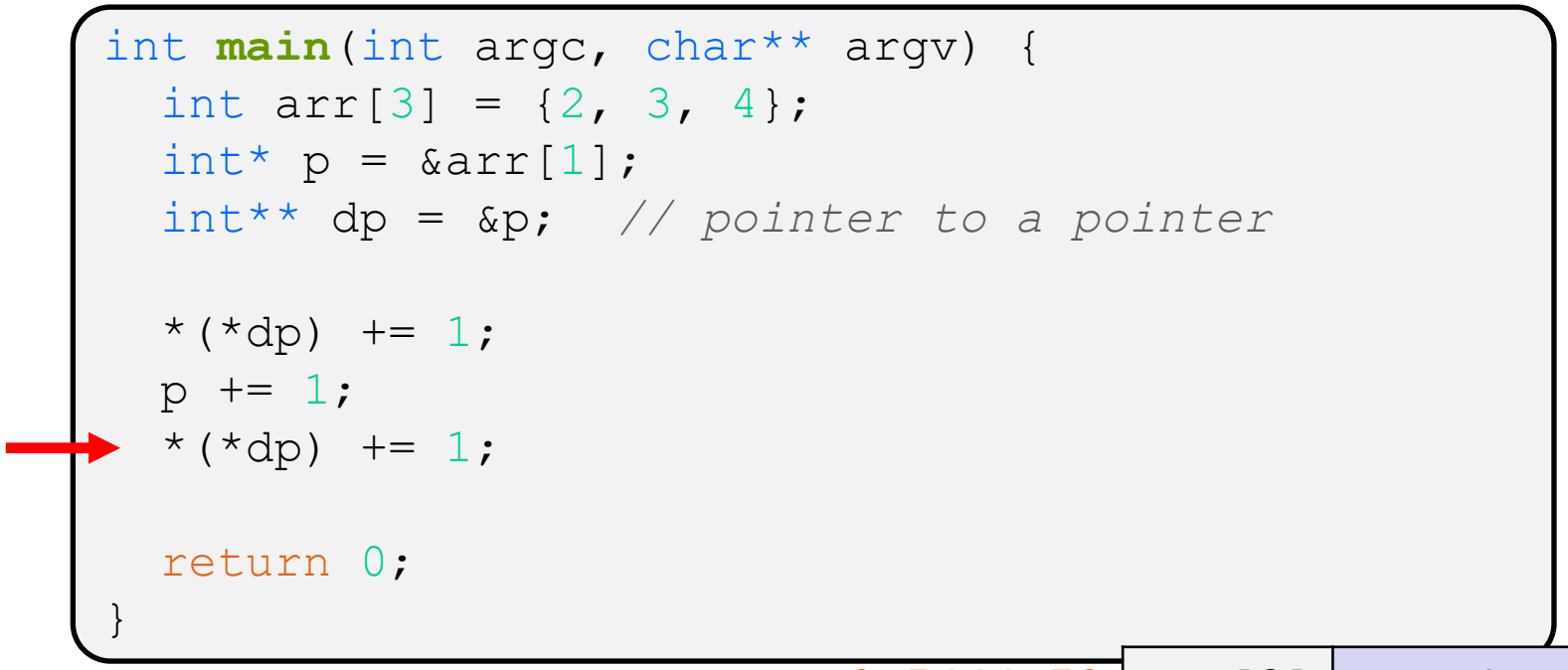


```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return 0;
}
```

Practice Solution (3/5)

Note: arrow points to *next* instruction to be executed.

boxarrow2.c

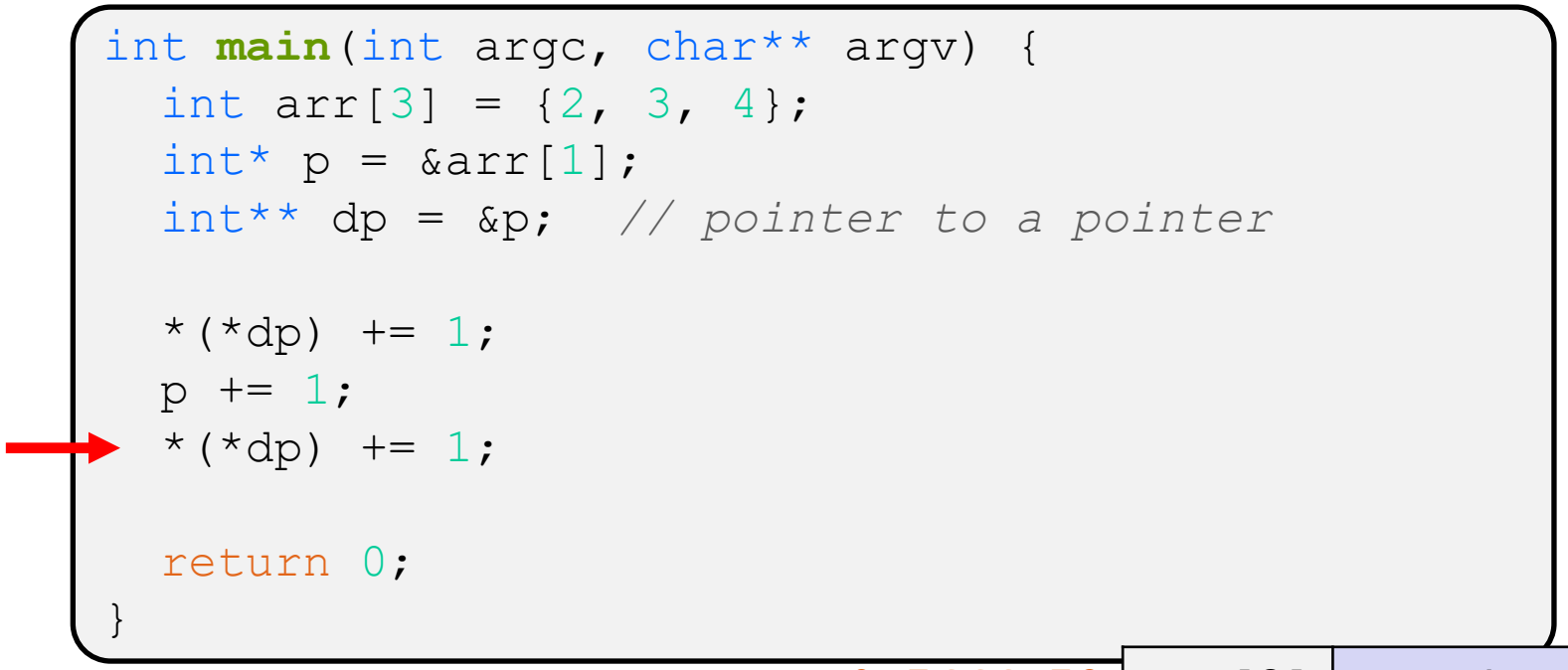


```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return 0;
}
```

Practice Solution (4/5)

Note: arrow points to *next* instruction to be executed.

boxarrow2.c

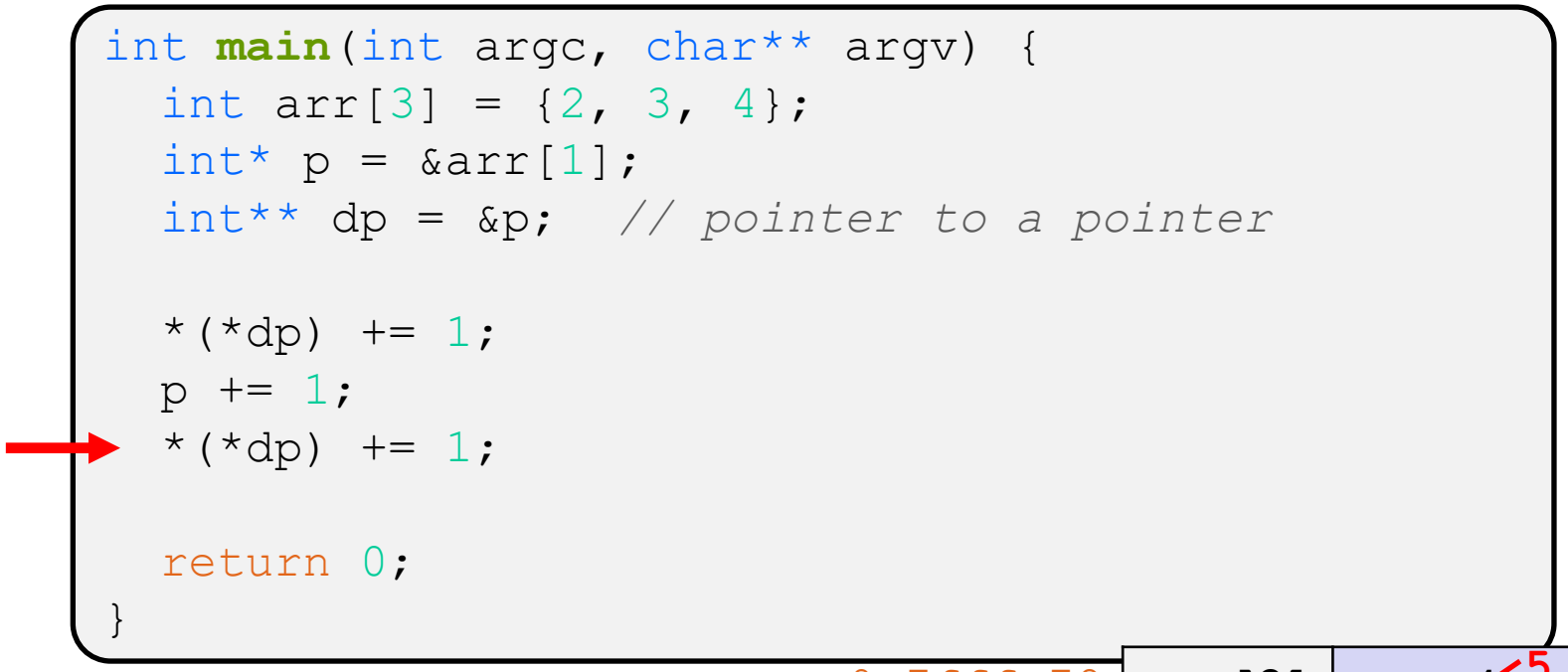


```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return 0;
}
```

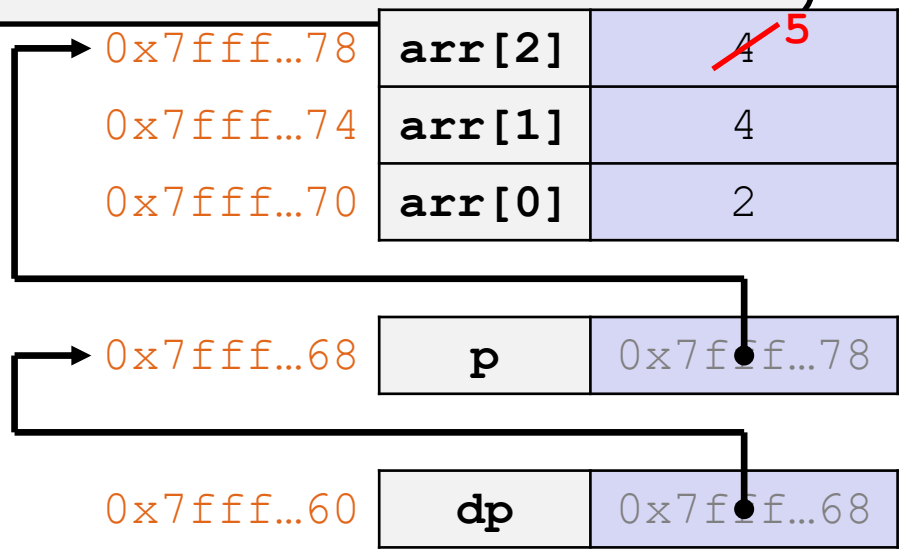
Practice Solution (5/5)

Note: arrow points to *next* instruction to be executed.

boxarrow2.c



```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int** dp = &p;  // pointer to a pointer

  *(*dp) += 1;
  p += 1;
  *(*dp) += 1;

  return 0;
}
```

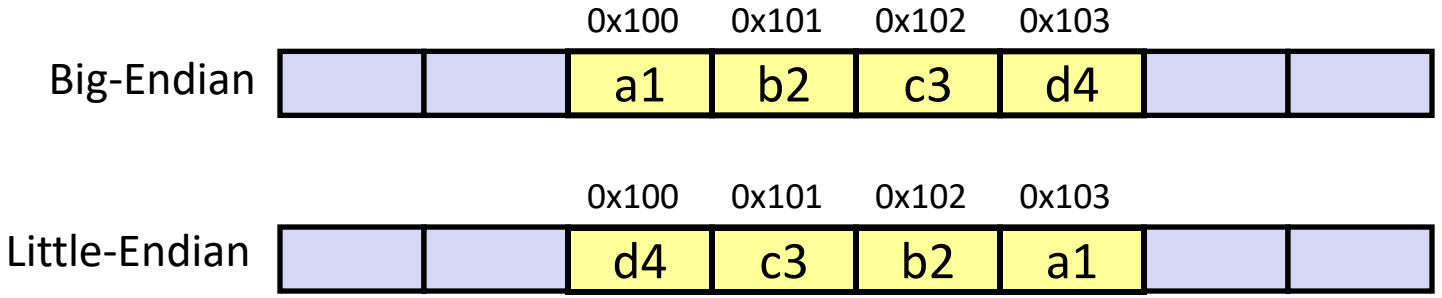


boxarrow2.c

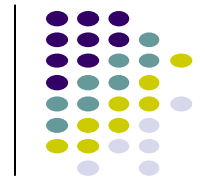
```c
#include <stdio.h>
#include <stdint.h>

int main(int argc, char** argv)
{

    int arr[3] = {2, 3, 4};
    int* p = &arr[1];
    int** dp = &p;  // pointer to a pointer

    *(*dp) += 1;
    p += 1;
    *(*dp) += 1;


    printf("&arr[2]: %p;  arr[2]: %d\n", &arr[2], arr[2]);
    printf("&arr[1]: %p;  arr[1]: %d\n", &arr[1], arr[1]);
    printf("&arr[0]: %p;  arr[0]: %d\n", &arr[0], arr[0]);

    printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
    printf("&dp: %p; dp: %p; *(*dp): %d\n", &dp, dp, *(*dp));

  return 0;
}
```

Endianness

- Memory is byte-addressed, so endianness determines what ordering that multi-byte data gets read and stored *in memory*
  - Big-endian: Least significant byte has *highest* address
  - Little-endian: Least significant byte has *lowest* address
    - X86-64
- **Example:** 4-byte data 0xa1b2c3d4 at address 0x100



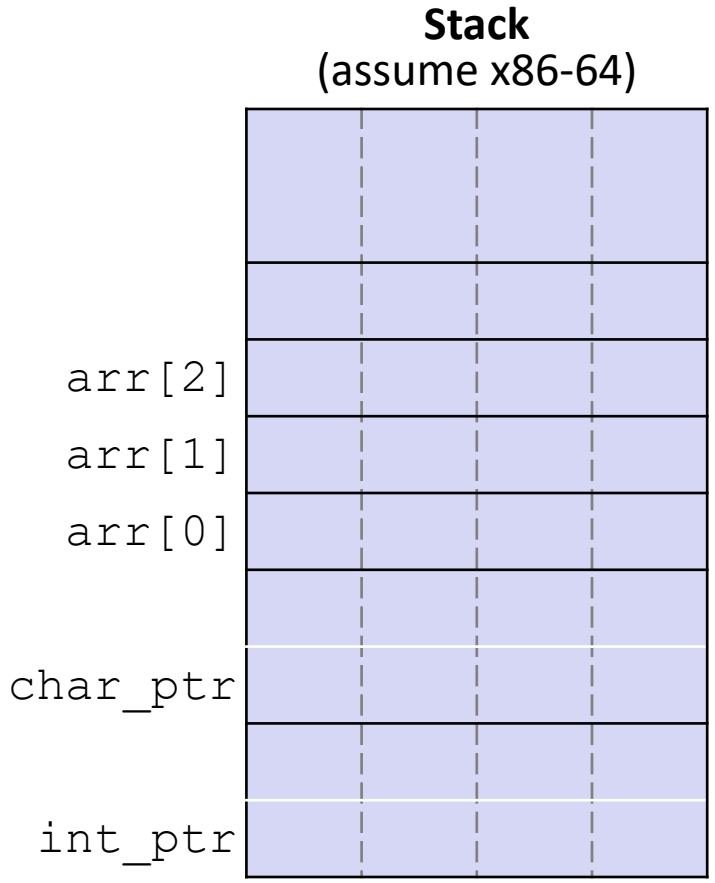
Pointer Arithmetic Example(1)



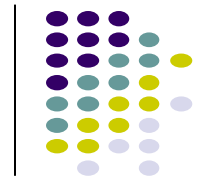
Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;  // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```

pointerarithmetic.c



Pointer Arithmetic Example(2)

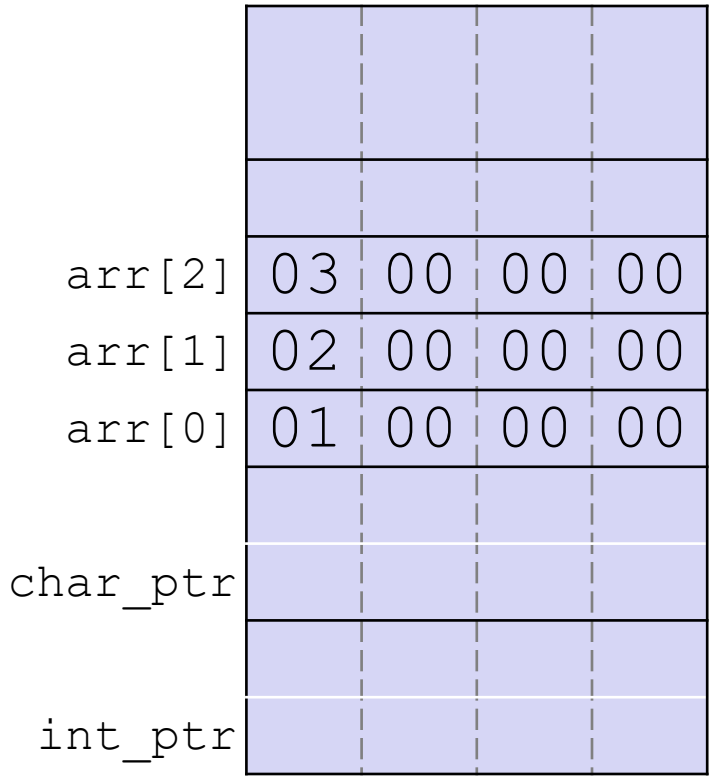


Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;  // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```

pointerarithmetic.c

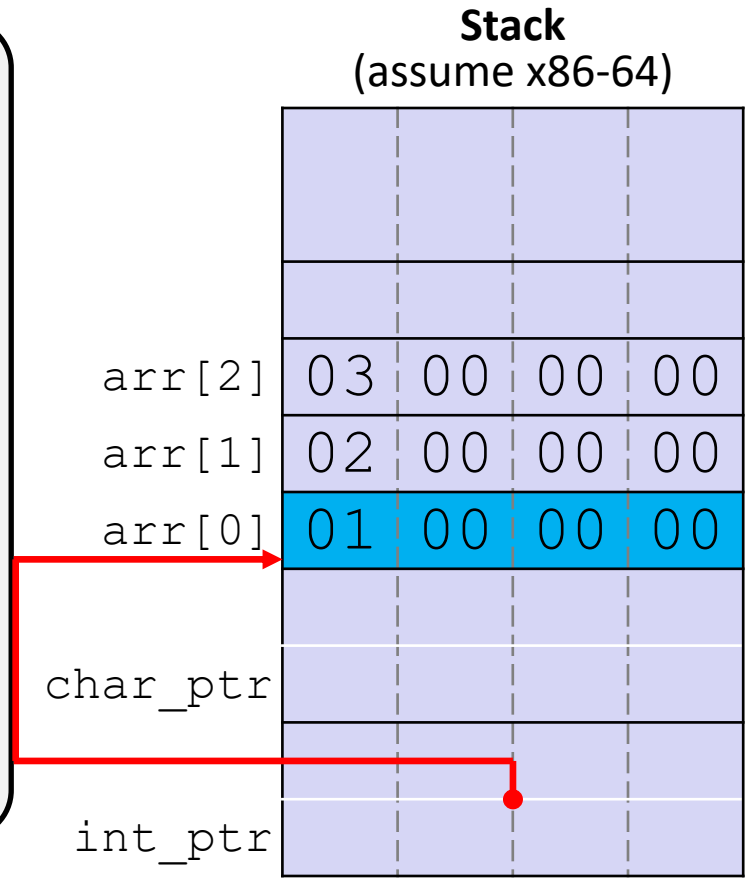
Stack
(assume x86-64)



Pointer Arithmetic Example(3)

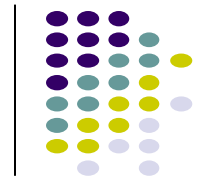
Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;  // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```



pointerarithmetic.c

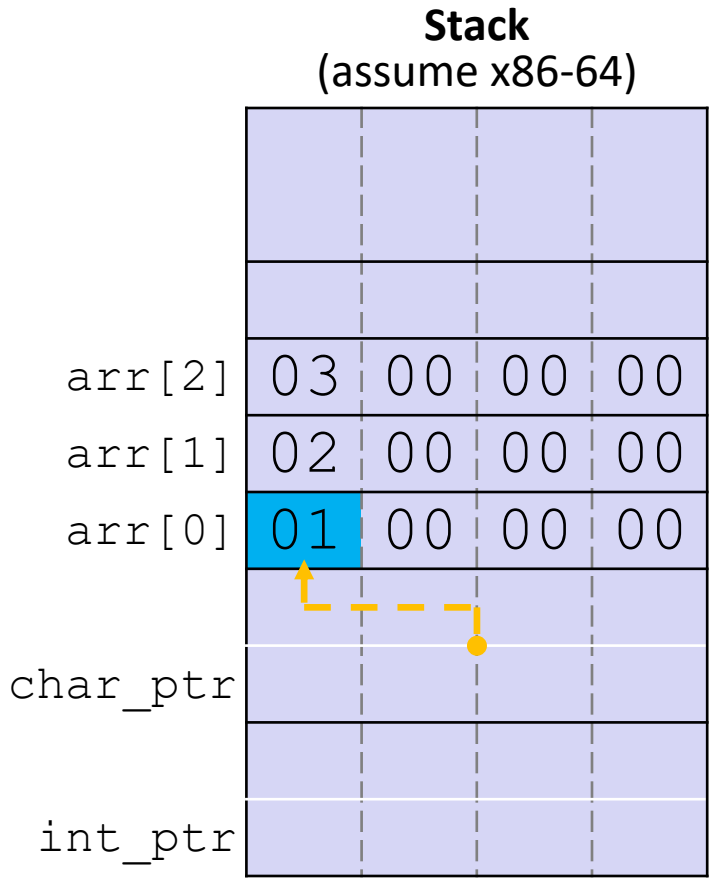
Pointer Arithmetic Example(4)



Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;  // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```

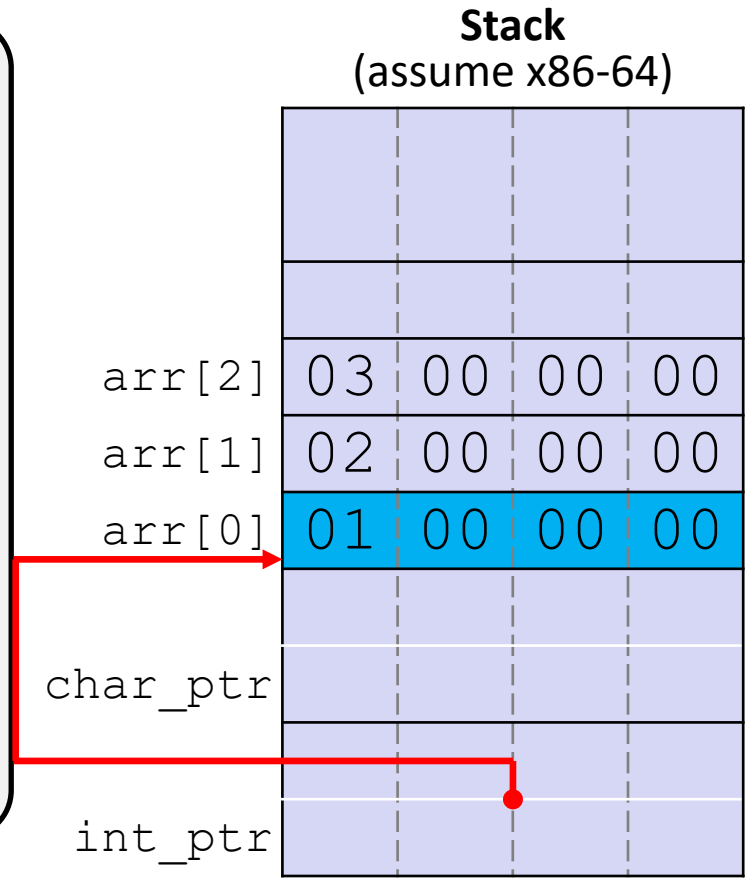
pointerarithmetic.c



Pointer Arithmetic Example(5)

Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;  // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```



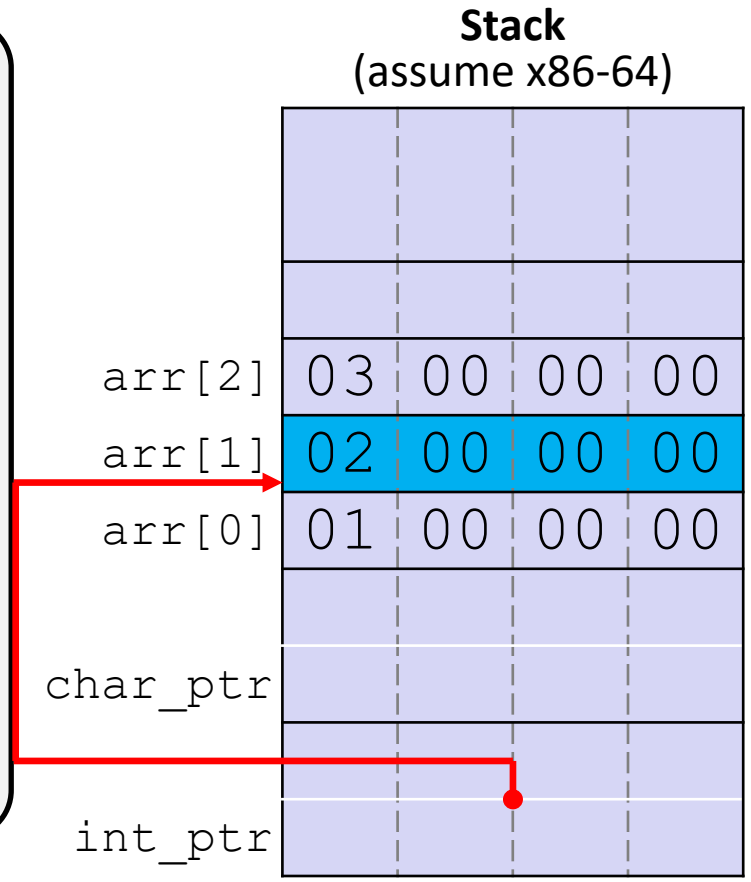
pointerarithmetic.c

```
  int_ptr:   0x0x7fffffde010
 *int_ptr:   1
```

Pointer Arithmetic Example(6)

Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;   // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```



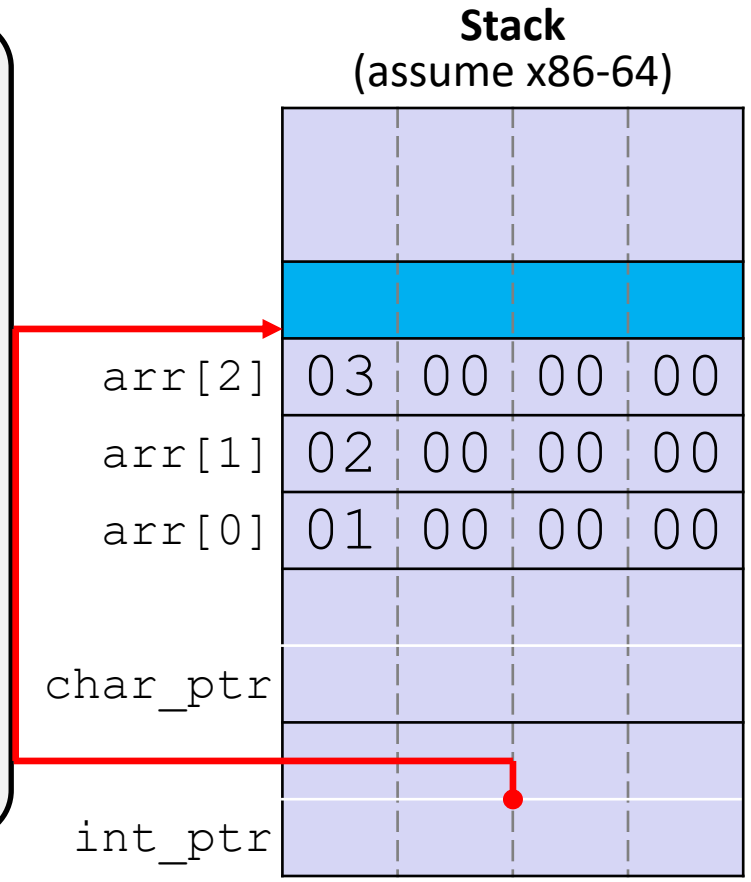
pointerarithmetic.c

```
  int_ptr:   0x0x7fffffde014
 *int_ptr:   2
```

Pointer Arithmetic Example(7)

Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;   // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```



pointerarithmetic.c

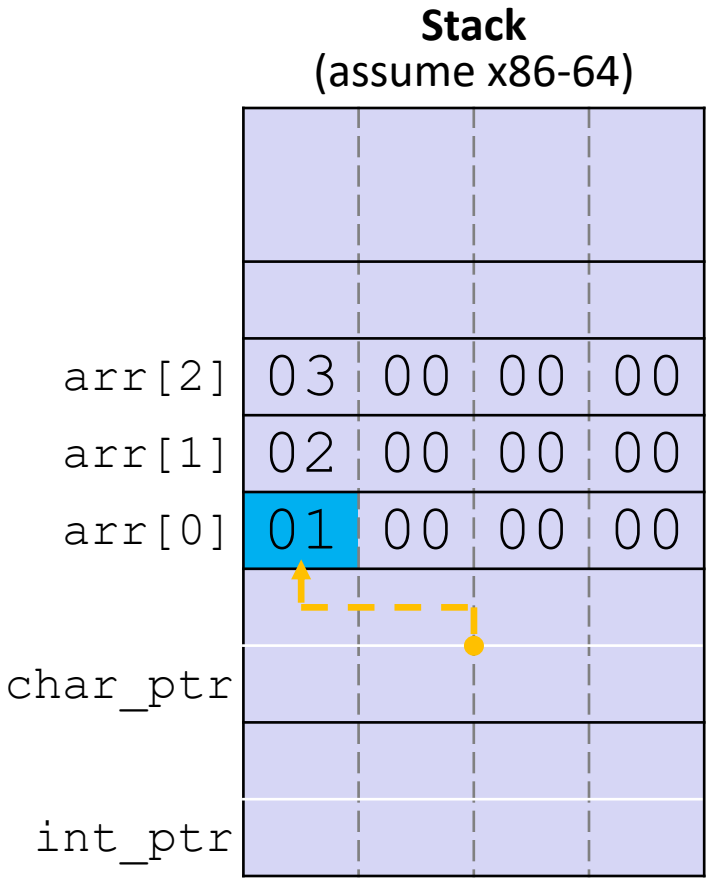
int_ptr: 0x0x7fffffde01**C**

***int_ptr:** **???**

Pointer Arithmetic Example(8)

Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;   // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```



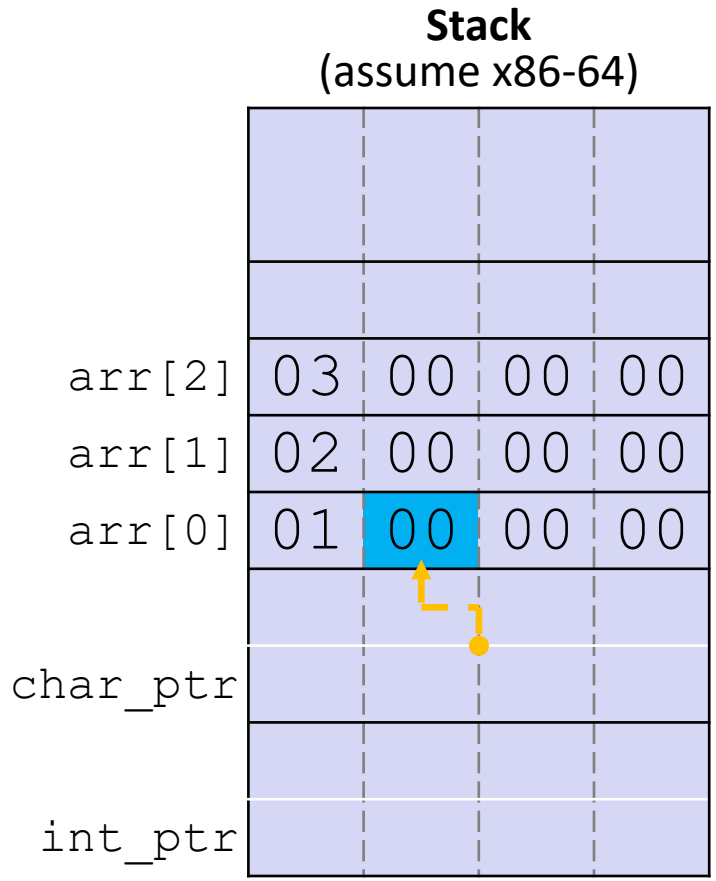
pointerarithmetic.c

```
 char_ptr:   0x0x7fffffde010
*char_ptr:   1
```

Pointer Arithmetic Example(9)

Note: Arrow points to *next* instruction.

```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;  // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```



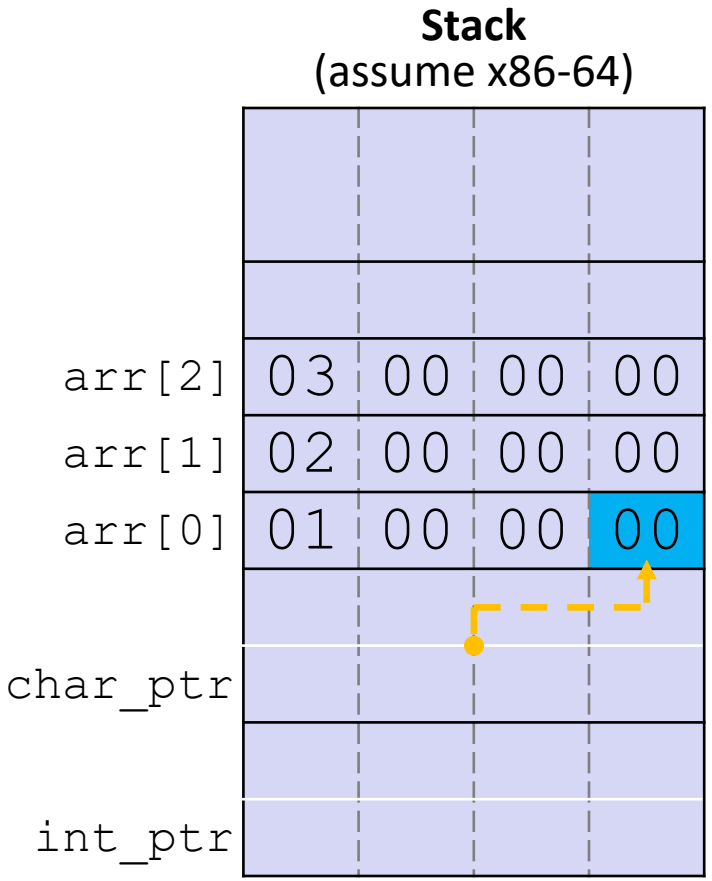
pointerarithmetic.c

```
 char_ptr:   0x0x7fffffde011
*char_ptr:   0
```

Pointer Arithmetic Example(10)

Note: Arrow points to *next* instruction.

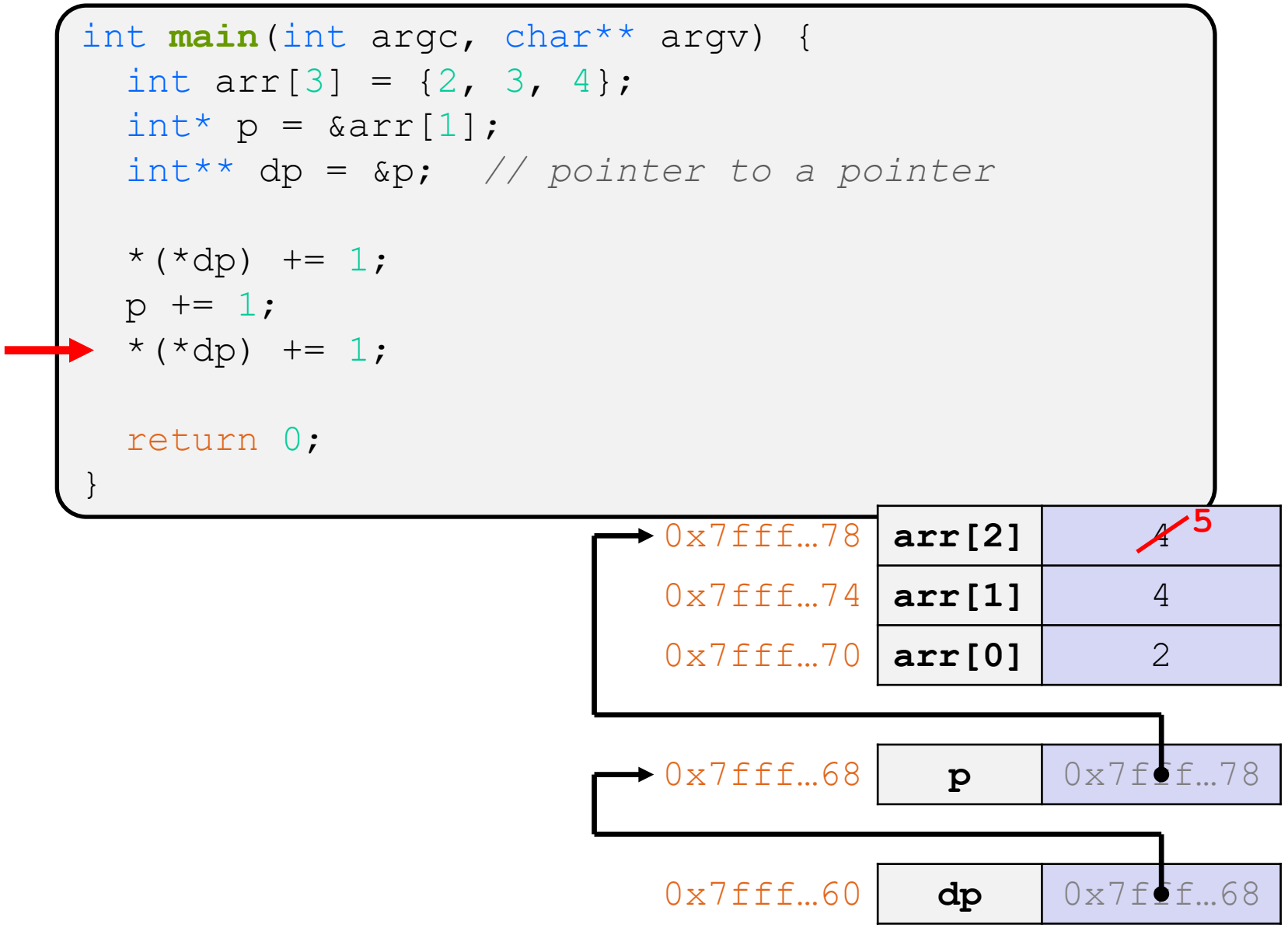
```
int main(int argc, char** argv) {
  int arr[3] = {1, 2, 3};
  int* int_ptr = &arr[0];
  char* char_ptr = (char*) int_ptr;

  int_ptr += 1;
  int_ptr += 2;  // uh oh

  char_ptr += 1;
  char_ptr += 2;

  return 0;
}
```



pointerarithmetic.c

```
 char_ptr:   0x0x7fffffde013
*char_ptr:   0
```

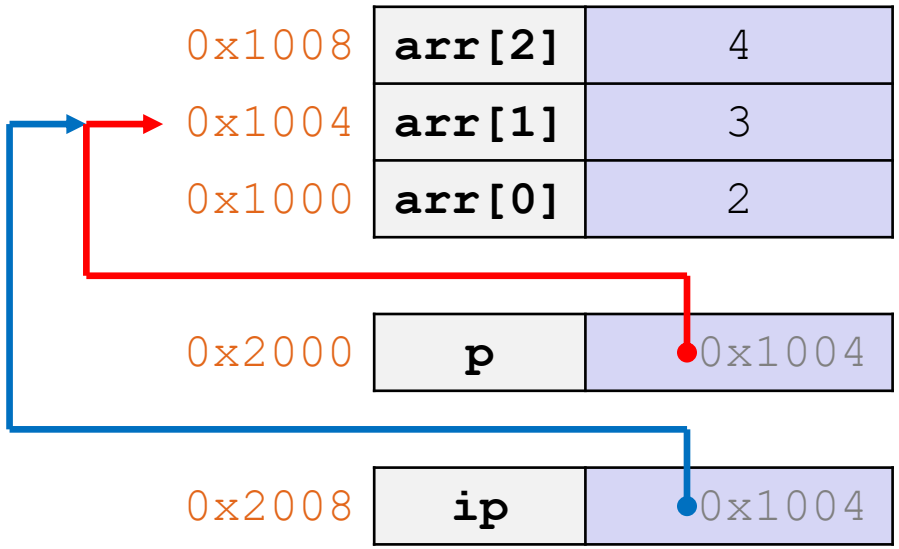
Comparison (1/2)



Comparison (2/2)

pointercomp.c

```
int main(int argc, char** argv) {
  int arr[3] = {2, 3, 4};
  int* p = &arr[1];
  int* ip = p;  // pointer assignment
  (*p) += 10;
  p += 1;
  printf("&p: %p; p: %p; *p: %d\n", &p, p, *p);
  (*ip) += 20;
  printf("&ip: %p; ip: %p; *ip: %d\n", &ip, ip, *ip);
}
```



Lecture Outline

- Pointers & Pointer Arithmetic
- **Pointers as Parameters**

C is Call-By-Value

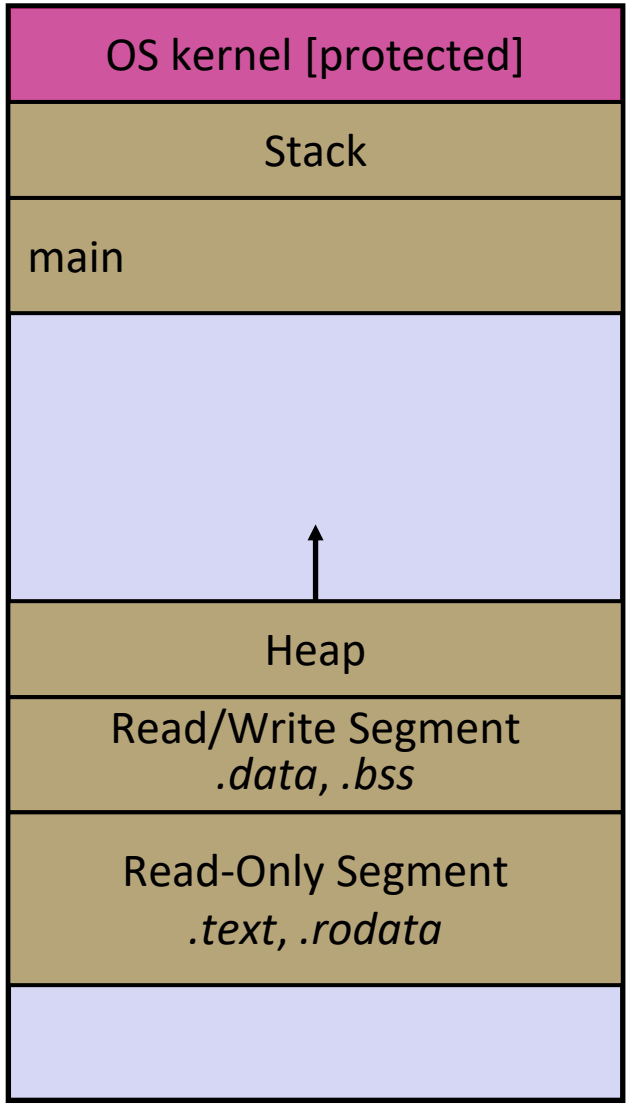
- C (and Java) pass arguments by *value*
  - Callee receives a **local copy** of the argument
    - Register or Stack
  - If the callee modifies a parameter, the caller's copy *isn't* modified

```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(a, b);
  ...
```

Broken Swap (1/7)

brokenswap.c

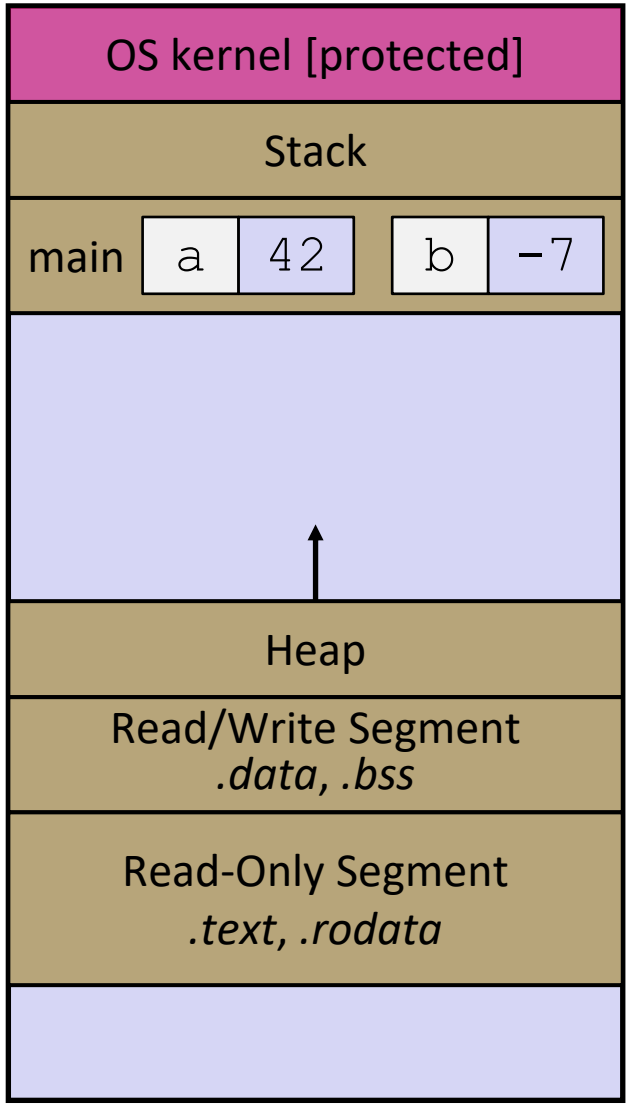
```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(a, b);
  ...
```



Broken Swap (2/7)

brokenswap.c

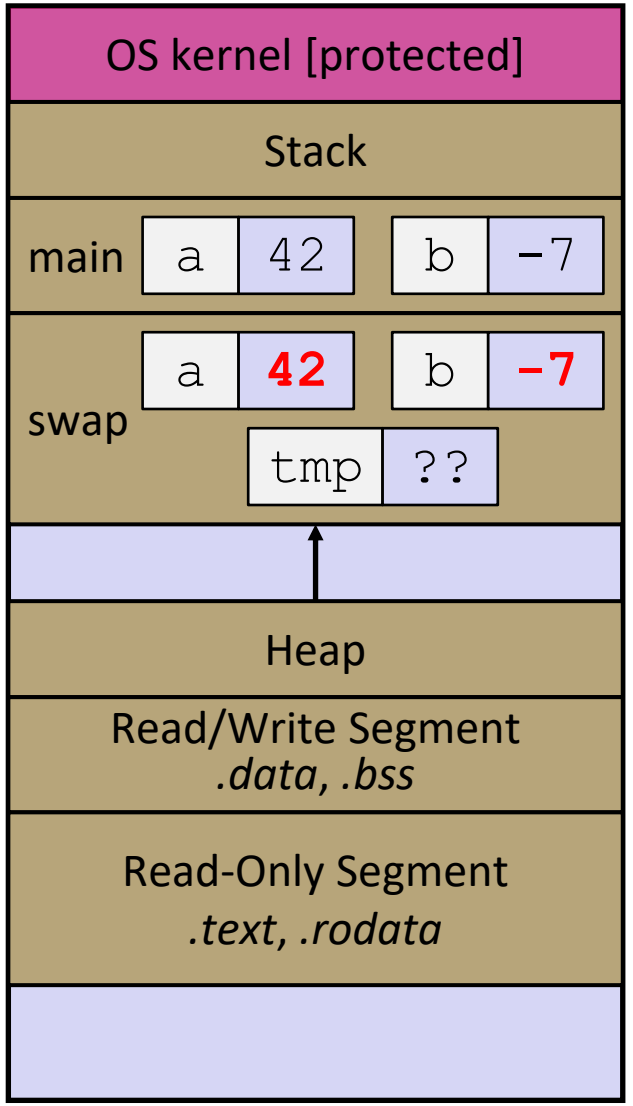
```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(a, b);
  ...
```



Broken Swap (3/7)

brokenswap.c

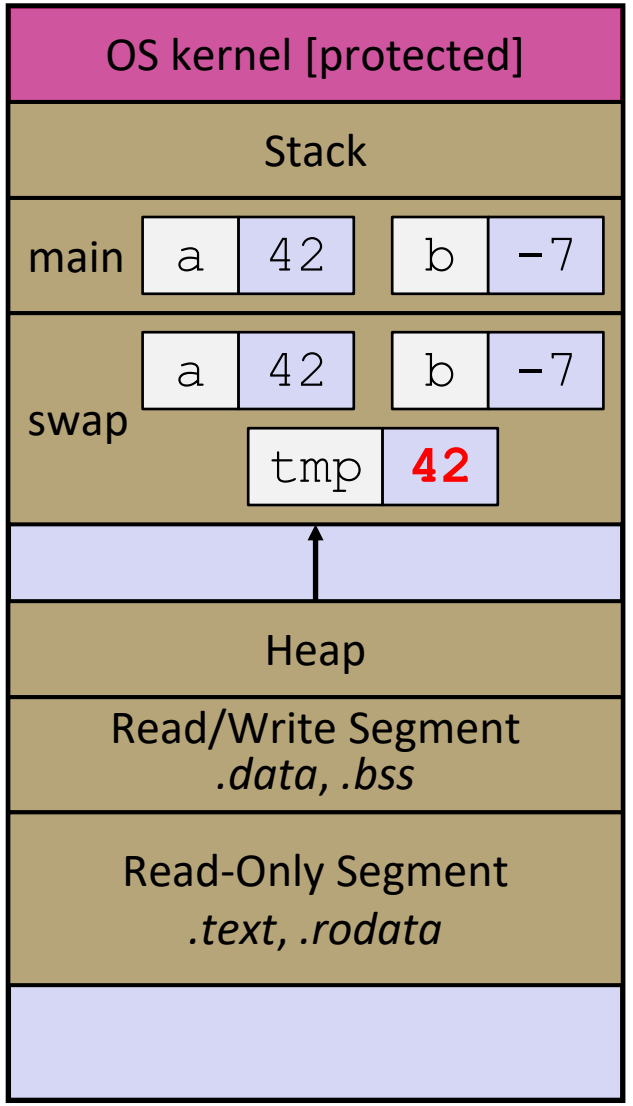
```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(a, b);
  ...
```



Broken Swap (4/7)

brokenswap.c

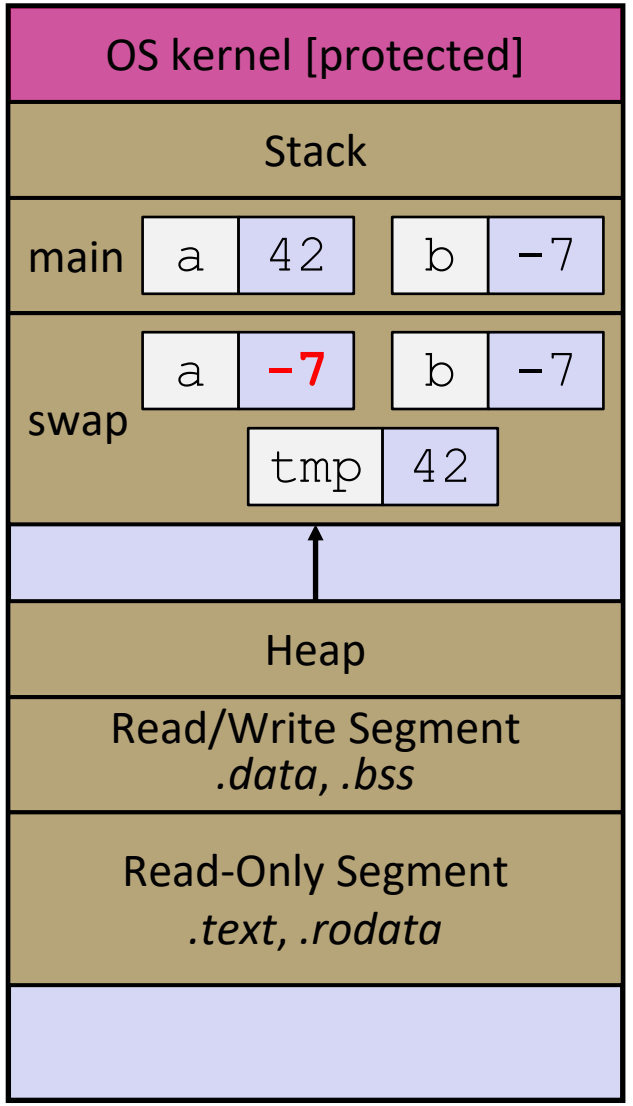
```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(a, b);
  ...
```



Broken Swap (5/7)

brokenswap.c

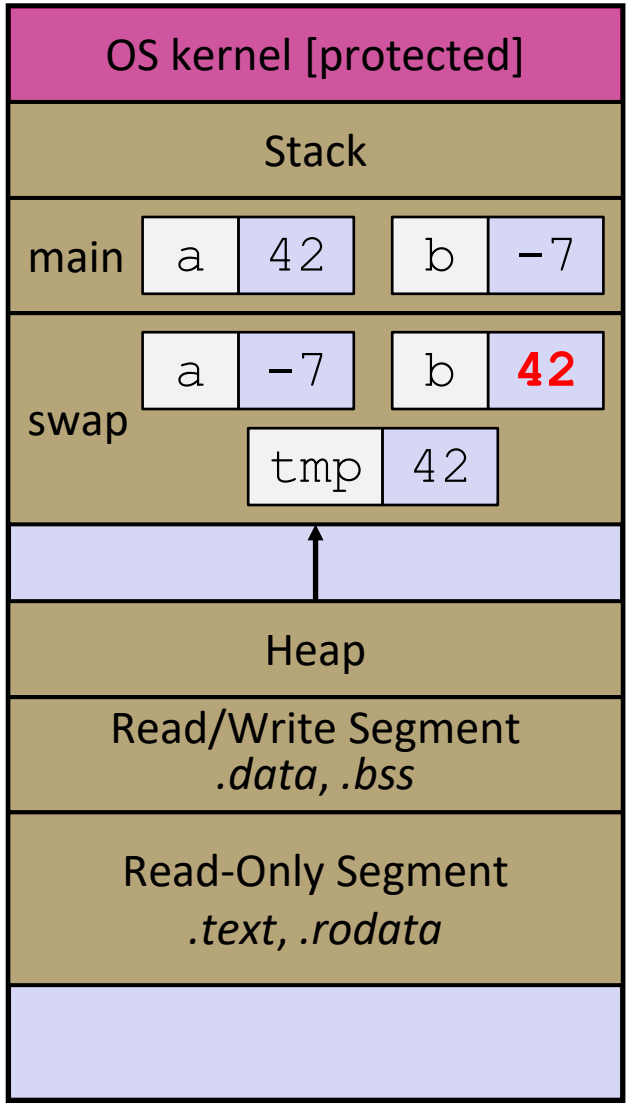
```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(a, b);
  ...
```



Broken Swap (6/7)

brokenswap.c

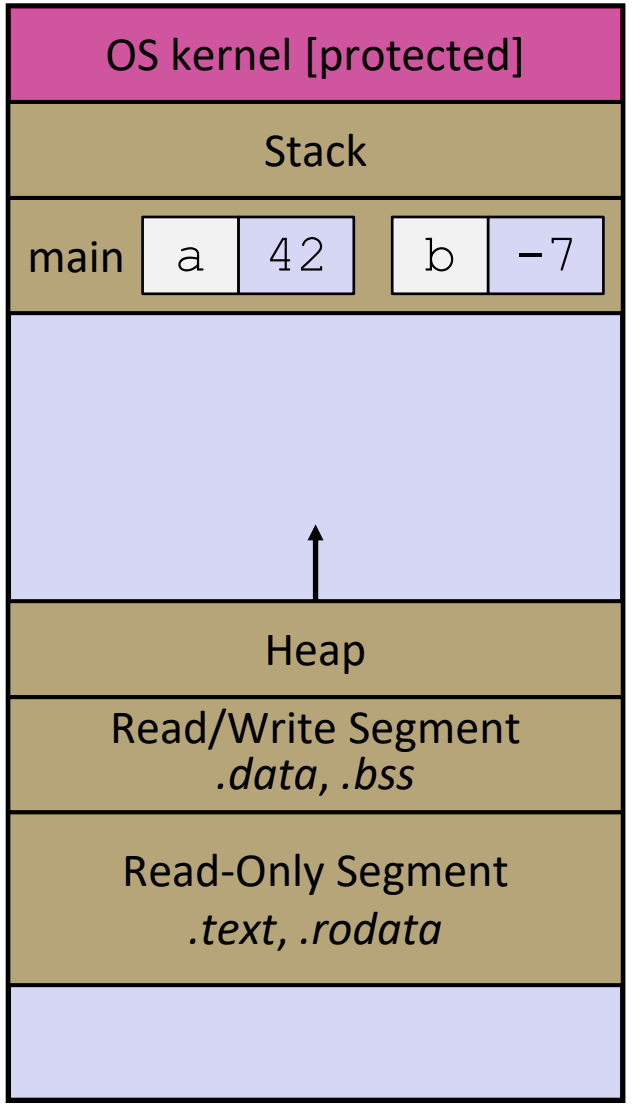
```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(a, b);
  ...
```



Broken Swap (7/7)

brokenswap.c

```
void swap(int a, int b) {
  int tmp = a;
  a = b;
  b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(a, b);
  ...
```



Faking Call-By-Reference in C

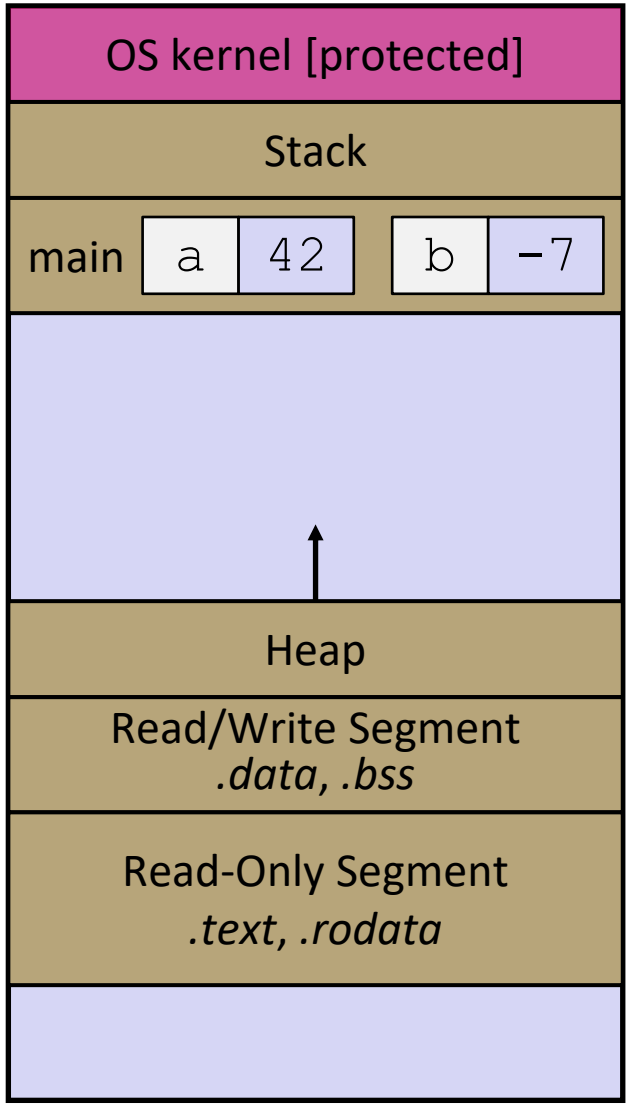
- Can use pointers to *approximate* call-by-reference
  - Callee still receives a **copy** of the pointer (*i.e.* call-by-value), but it can modify something in the caller's scope by dereferencing the pointer parameter

```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(&a, &b);
  ...
```

Fixed Swap (1/6)

swap.c

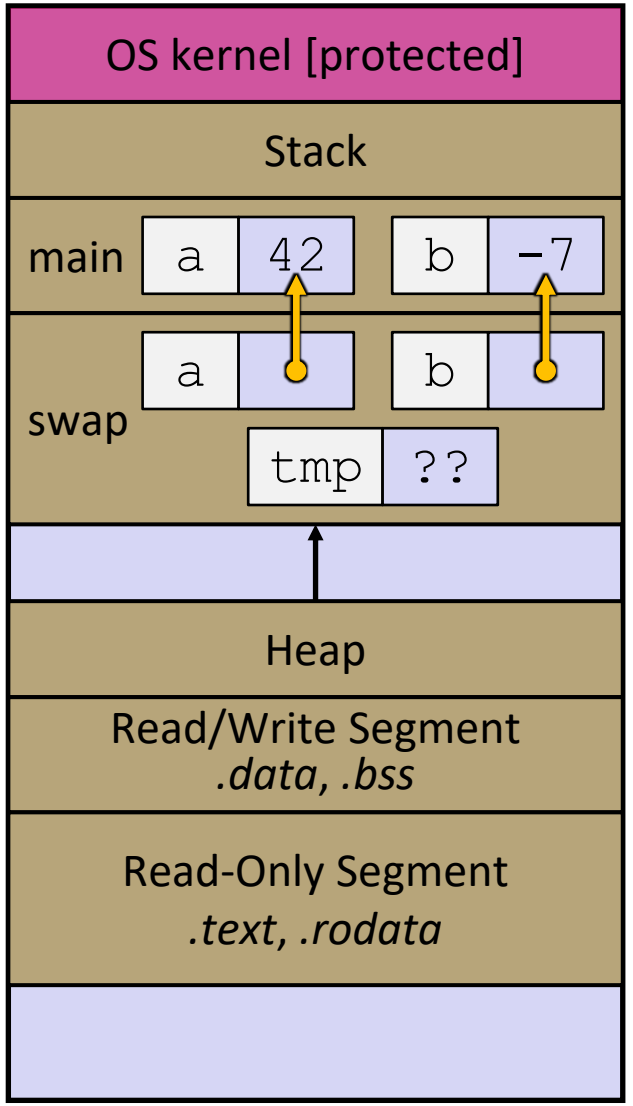
```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(&a, &b);
  ...
```



Fixed Swap (2/6)

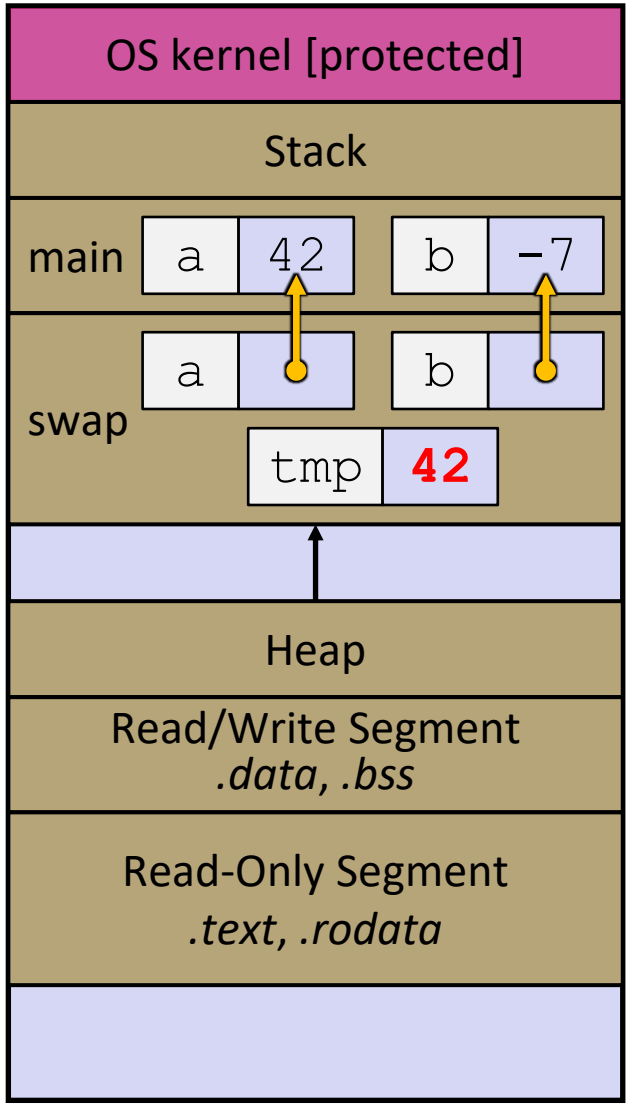
swap.c

```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(&a, &b);
  ...
```



Fixed Swap (3/6)

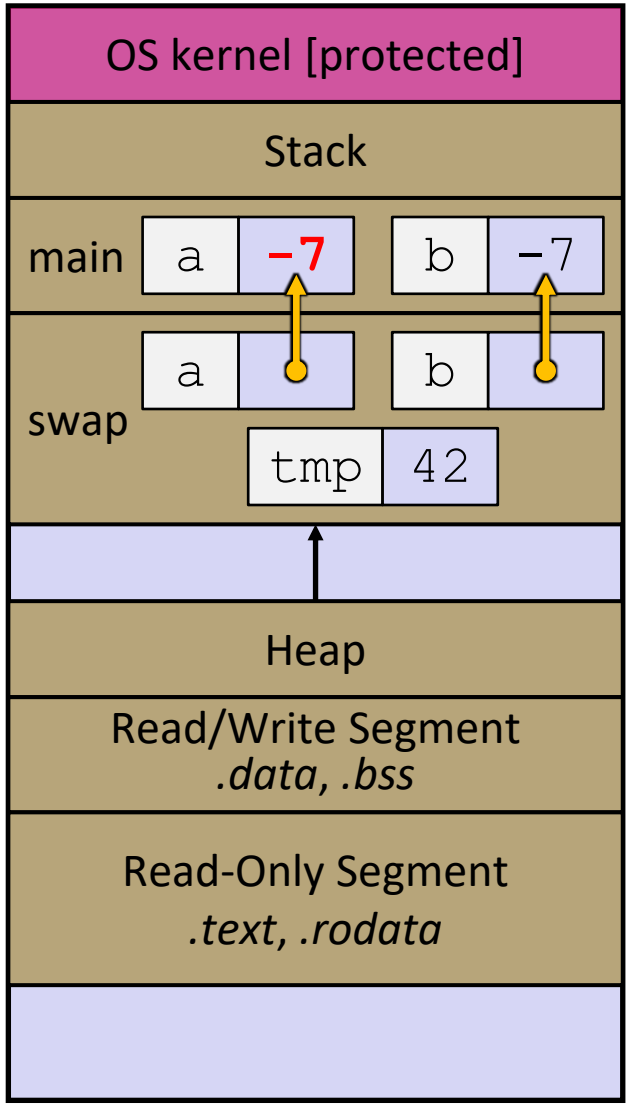
```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(&a, &b);
  ...
```



Fixed Swap (4/6)

swap.c

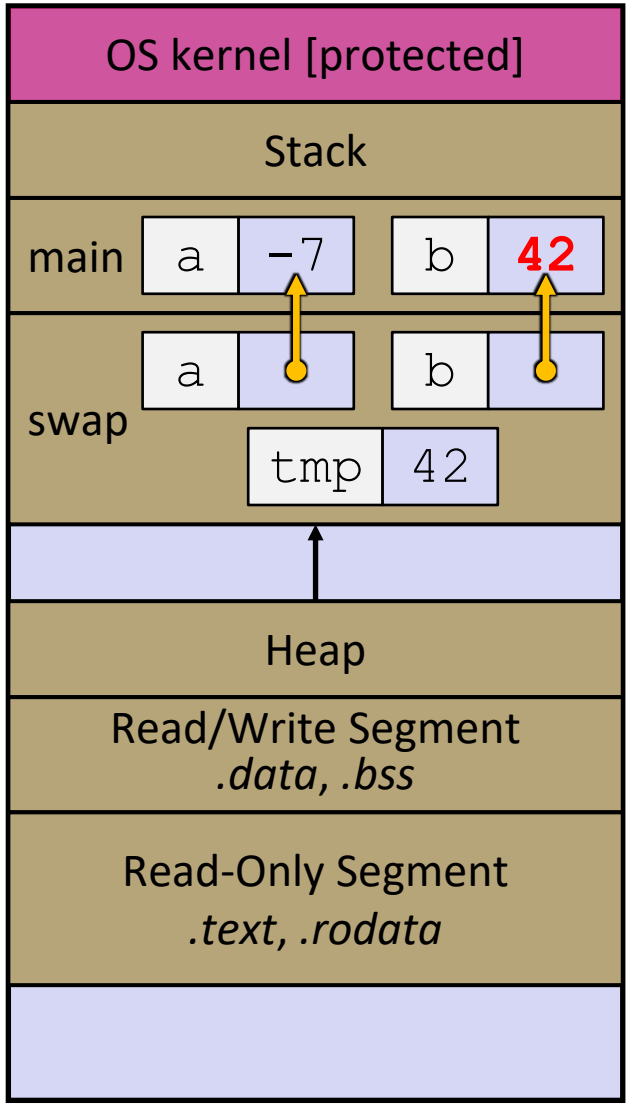
```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(&a, &b);
  ...
```



Fixed Swap (5/6)

swap.c

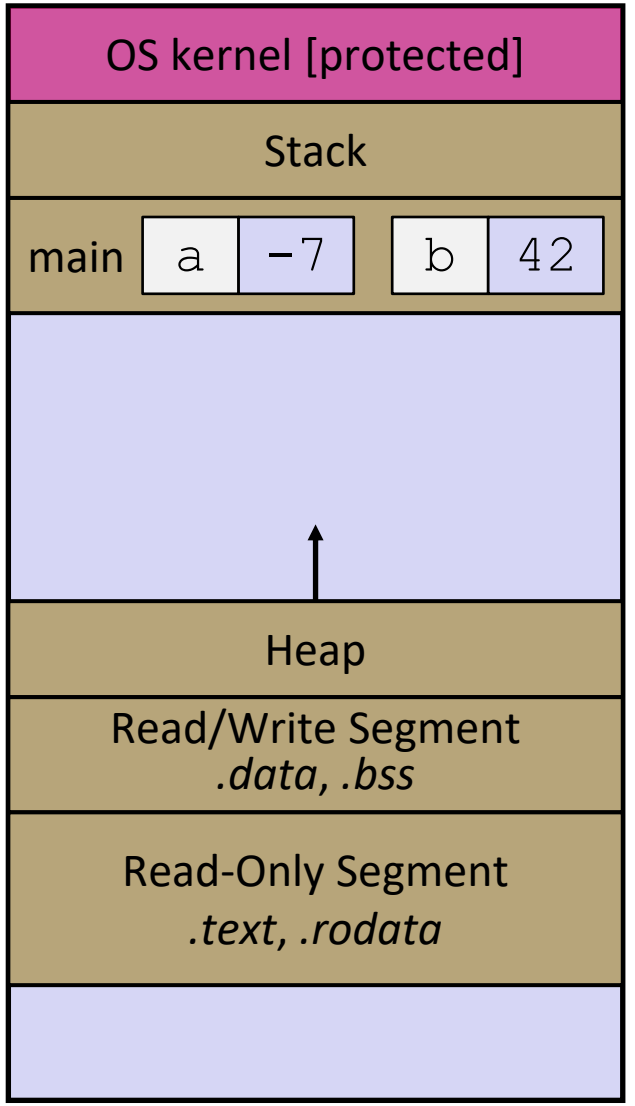
```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(&a, &b);
  ...
```



Fixed Swap (6/6)

swap.c

```
void swap(int* a, int* b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main(int argc, char** argv) {
  int a = 42, b = -7;
  swap(&a, &b);
  ...
```



Q&A